Chapter 17

Beyond the Interface

(with Meeko Oishi and Claire Tomlin)

The previous chapter was about a systematic methodology for verification of interfaces, making sure that the interfaces are correct for the task. We now take it one step further—analyzing and verifying the underlying system. I will use the same presentation style as before: illustrating a situation that we are all familiar with, and then moving on to analyze a more complex automatic system.

Red Light Violation

Consider the following scenario: you're driving toward an intersection, and you can see the green traffic lights looming in the near distance. But as you approach the intersection, the light turns yellow. What do you do? Should you stop, or perhaps drive through? As we all know, the answer depends on many factors as well as the context of the situation—the car's distance and speed, whether there's a car behind, if you're in a hurry, if there are cars waiting at the intersection; or perhaps this is the middle of the night and the intersection is empty and nobody is looking. Let's begin by focusing on two main issues: the distance from the intersection and the car's speed.

And to sharpen the issues even more, consider here an intersection that has a video camera and a large sign that says "Red light violation, $281 minimum fine."

First we analyze the two available options for the driver: braking before the intersection or proceeding through the intersection. Figure 17.1(a) shows a region from which braking is safe. Given any combination of speed and distance within the dark gray region, the driver can always safely brake. But how do we know that? We know the car must stop at the white line before the intersection; at that point, the speed is 0. From there we go backward and

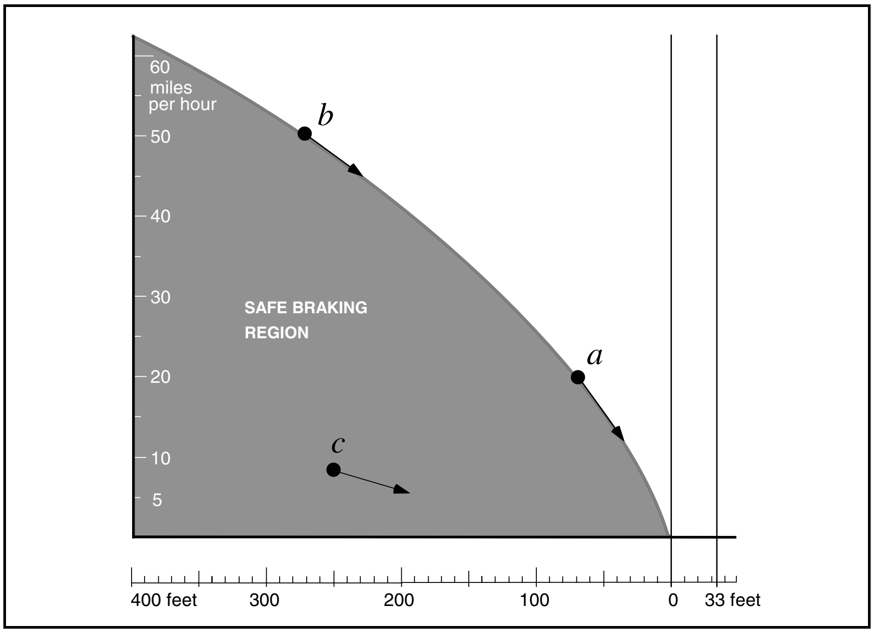

Figure 17.1 (a). The safe breaking region. (The horizontal axis shows distance, in feet, from the intersection and the vertical axis shows the car's speed in miles per hour.)

calculate, based on the car's maximum braking performance, the stopping distance for every speed (from 1 to 60 miles per hour).

The yellow light is 4 seconds long, and we take into account that it takes the (average) driver 1.5 seconds to react and press on the brakes. So, for example, at point *a*, the car's speed is 20 miles per hour and if you apply maximum braking, you *will* stop at the white line; the same is true when you apply maximum braking at point *b* (speed 50 miles per hour). Naturally, for every point inside the safe-braking region, for example point *c*, maximum braking is not required—the driver can brake lightly and still be able to stop before the intersection.

Now we turn our attention to the other option—driving through the intersection. Let us assume for now that the driver, once he or she observes the yellow light, simply maintains current speed. How do we figure out the safe region for driving through? Just as before, we start from the intersection and work our way back.

Contrary to common belief, it is *legal* to be caught inside the intersection when the light turns red and cross to the other side with the red light above you. What is *illegal*, however, is to enter the intersection when the light is red (see the discussion in the endnotes about California's driving regulations). Therefore, we calculate from the white line at the entry of the intersection,

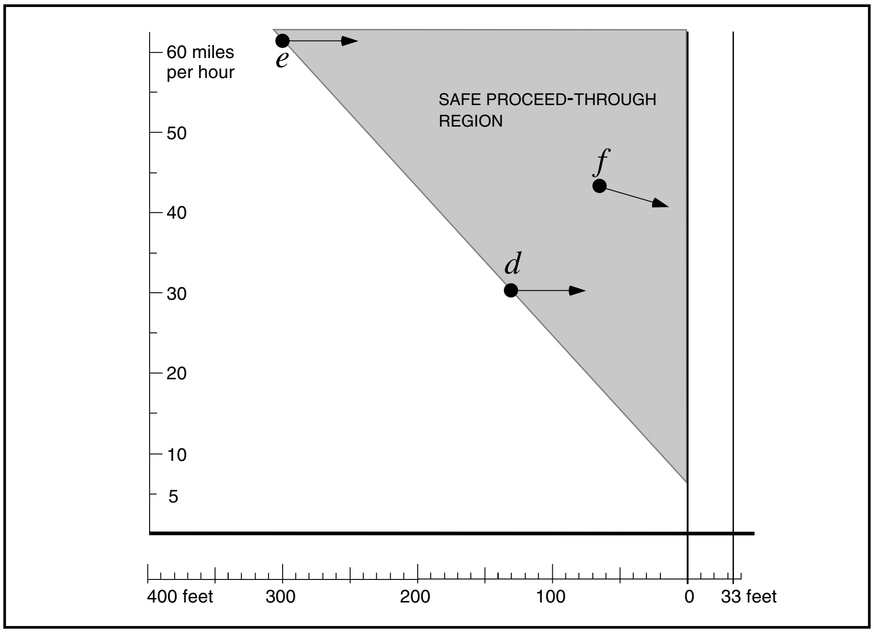

Figure 17.1 (b). Safe proceed-through region.

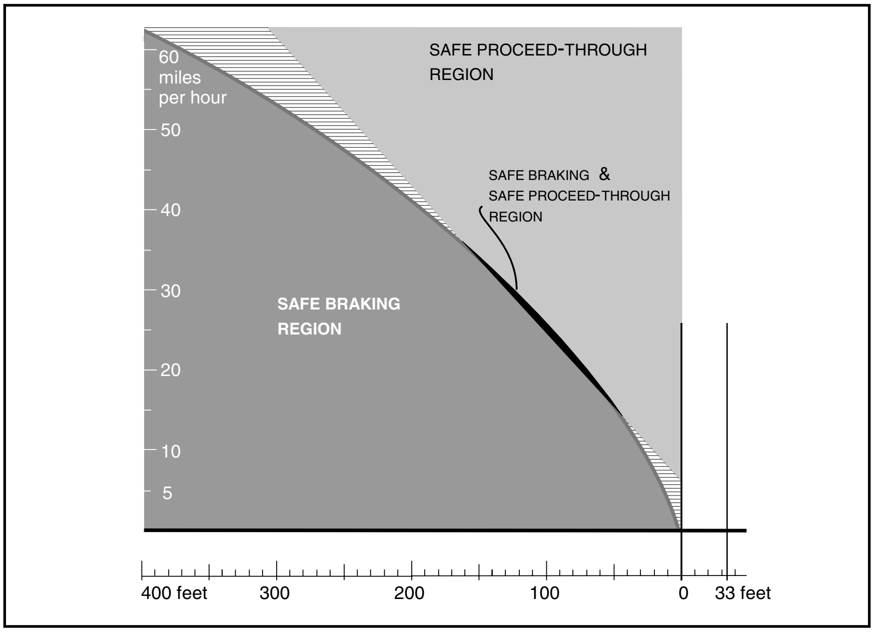

Figure 17.1 (c). Composite of the safe braking and safe proceed-through regions.

assuming that we want to pass the white line at a minimum speed of 7 miles per hour (because we do need to vacate the intersection before it turns green for the crossing traffic). In figure 17.1(b), we see that when the speed is 30 miles per hour and the distance from the intersection is 130 feet (point *d*), we can safely drive through. From point *e* (speed 62 miles per hour, distance 300 feet) we can also drive through and pass the intersection in time. For every point inside the safe proceed-through region (e.g., point *f*), maintaining the current speed is not required—the driver can even reduce speed and still cross the intersection before the light turns red.

Composite Graph

Now that we know the consequences of either stopping before the intersection or driving through it, we can turn to the decision itself. How do we know, when the yellow light comes on, whether we should brake or proceed through? To analyze this dilemma, specifically in situations where it is not so obvious which option is safe and legal, we need to consider the "safe braking" region and the "safe proceed-through" region together.

The composite shown in figure 17.1(c) divides the entire operational region into four sub-regions. The dark gray region is the "safe braking," the light gray is the "safe proceed-through," and the narrow black region is where they overlap. If you are in the black region, you can either brake or proceed through, and you'll still be safe and legal.

But what are the hatched areas?

These are sub-regions of the operational space from which you can't safely brake or proceed through the intersection. Here's the problem: if you stop, even at maximum braking, you will find yourself entering the intersection in red; if you proceed-through, the light will turn red before you reach the intersection. The point is that given the two options, brake or drive at constant speed, there is just nothing you can do. If you are in the hatched region when the light turns yellow, you will commit a violation and get a ticket.

So now you're thinking to yourself, wait a minute, what about "gunning" through the intersection. Won't we avoid the red light? Well, we all know that accelerating to cross an intersection that is about to turn red can be rather dangerous, but regardless, let's consider this acceleration option. In figure 17.2, we expand the "proceed through" region to include the acceleration option. If the driver is in the white region, for example at point g, and he or she accelerates, it is possible to pass through and vacate the intersection in time.

Note that by accounting for accelerations, the hatched regions in figure 17.2 have shrunk, but they were not eliminated. Therefore, regardless what we do, there are still the regions where we are just too fast to stop in time and too far away to accelerate through the intersection. The point is that there are

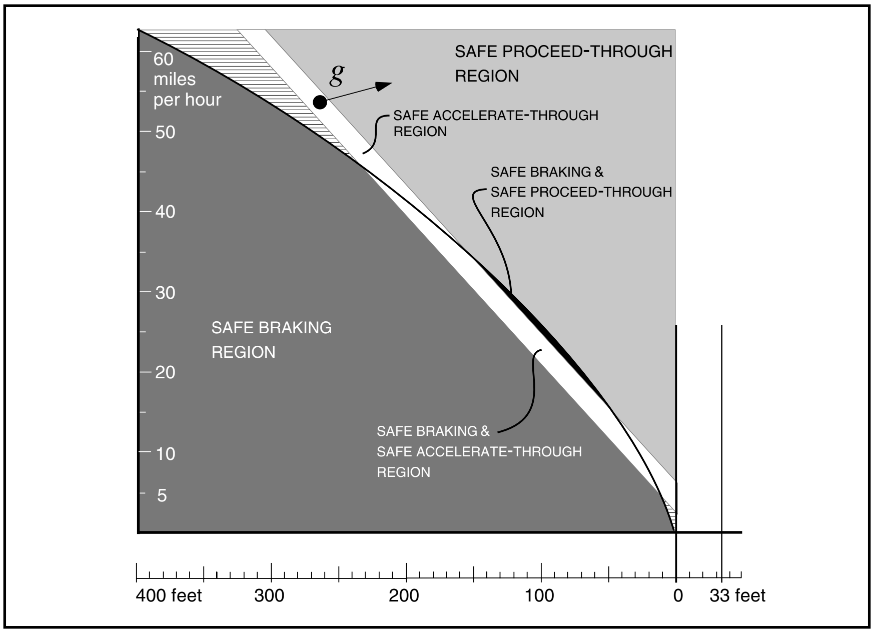

Figure 17.2. With acceleration.

combinations of speed and distance from which it is *impossible* to be safe. This is one of those "damned if you do and damned if you don't" situations. The hatched regions represent a lock, because if the driver is there when the light turns yellow—there is no way to escape.

Possible Remedies

The above problem is called the "yellow interval dilemma" and is well known to traffic engineers. The unsafe (hatched) region at the top of the graph is called "the dilemma zone." Now let's consider some of the viable design options for eliminating the unsafe regions. For a start, we can limit the speed to 35 miles per hour (see figure 17.1[c]). That way nobody will reach the unsafe (hatched) region at the upper end of the curve; and if they did, it is because they were speeding (now if you are caught you may get two tickets—one for speeding and one for a red light violation). But there are problems with this speed limit option because sometimes, for example outside of town, 35 miles per hour is just too slow.

Another option is to increase the duration (interval) of the yellow light, which will increase the size of the "proceed through" region and eliminate the unsafe region. Interestingly enough, it turns out that this solution is only partially successful, and the reason for this is due to the human factor: once drivers notice that there is ample time, they tend to ignore the yellow light and proceed through the intersection even when they are far away from it. Short of increasing the yellow light interval, one way that traffic engineers deal with the problem, albeit not directly, is to delay the green light to the crossing traffic. That is, for about two seconds, all sides of the intersection are red. The solution increases overall safety by adding a buffer zone, but does not eliminate the yellow light problem (and the unwarranted red-light violations).

Automatic Landing System

In the example above we analyzed the intersection and the car's performance to understand possible actions on the part of the driver; but in an implicit way, it was also a verification of the yellow light interval. By creating a composite of two graphs, we identified regions that allow the driver safe braking and safe passage and regions that don't. The same approach can be used for analyzing and verifying more complex systems—in particular, automated control systems, where we want to verify the design of the automation itself as well as human interaction with it.

In most automated control systems, and in particular those that are used in safety-critical systems, the user has the authority to break away or disengage the automation when unsafe situations arise. One case in point is the automatic landing system in modern airliners. As the name implies, this system flies the approach, makes the landing, and then steers the aircraft to the end of the runway. Automatic landing systems are commonly used in bad weather, specifically, in a condition called "zero-zero" in aviation lingo, which means that the out-of-the-cockpit visibility is zero and the clouds or fog reach all the way to the ground.

In these severe visibility conditions, only the automatic landing system is permitted to make a landing. Therefore, automatic landing systems are designed to be extremely accurate and have built-in redundancies to reduce the likelihood of a system malfunction. Furthermore, automatic landing systems go through rigorous testing and retesting on a monthly basis. It is the only component of the automatic flight control system for which we cannot rely on the pilot as a backup in case the system is in error. Why? Because in zero visibility the pilot cannot see the runway at all, and therefore he or she cannot judge if the autoland system is doing its job properly. Nevertheless, the pilots are required to monitor the automatic landing system, making sure that it switches among its internal modes properly; that it maintains appropriate speed and pitch attitude; and is tracking the glide-slope (which the pilots can see on their displays). In making an automatic landing, the autopilot is in full control of the situation. But there is one option that is always available to the pilot—to discontinue the approach and make a go-around.

A go-around is a maneuver that every seasoned airline passenger has experienced at one time or another. During the approach for landing, all of a sudden the aircraft pitches up and the engines roar, and within a few seconds the aircraft begins to climb rapidly. Usually the reassuring voice of the captain comes on the public-address system and informs us that we are "going around" for yet another approach and landing. From the passengers' point of view, the go-around is perhaps not a very pleasant experience. On the pilots' side, go-arounds are well-practiced maneuvers with the intention of taking the aircraft away from an unsafe landing. Sometimes, a go-around is requested by air traffic control—perhaps the aircraft has come too close to another aircraft on the approach, or there is debris, a vehicle, or another aircraft on the runway.

Going Around

In modern automated aircraft, the pilots execute a go-around by pushing a small lever located on the throttles. From then on, the automatic flight control system does the rest—pitching up the aircraft, advancing the engines to full thrust, and flying the aircraft to a higher altitude (usually 2,000 feet). From our point of view, a go-around is a disturbance event because it may come *any time* during the approach. Performing a go-around places considerable workload on the pilots: first, it is usually a surprise and they need to break quickly away from their focus on landing. From there on, flaps must be set, landing gear raised, a reference airspeed maintained, and checklists performed. Sometimes the pilots also need to formulate new plans and make quick decisions, then consult charts and enter information into the computer (not to mention briefing the weary passengers).

Although the pilots can always disengage the autopilot and execute the go-around manually, it is recommended, and in some airlines mandatory, that this go-around maneuver be performed automatically. The reason is that the maneuver is quite difficult to perform optimally, that is, with a minimum loss of altitude, especially when the aircraft is close to the ground.

Figure 17.3 shows the sequence of mode switching in making an automatic landing and the consequence of engaging the go-around mode: Initially, the AUTOLAND mode is engaged and the airplane is flown automatically toward the runway. At 50 feet above the runway, the system transitions to FLARE mode and begins the curved maneuver to touchdown. After the airplane lands, the system goes into ROLLOUT mode and steers the airplane on the runway's centerline until the pilots bring the airplane to a stop. A go-around may be

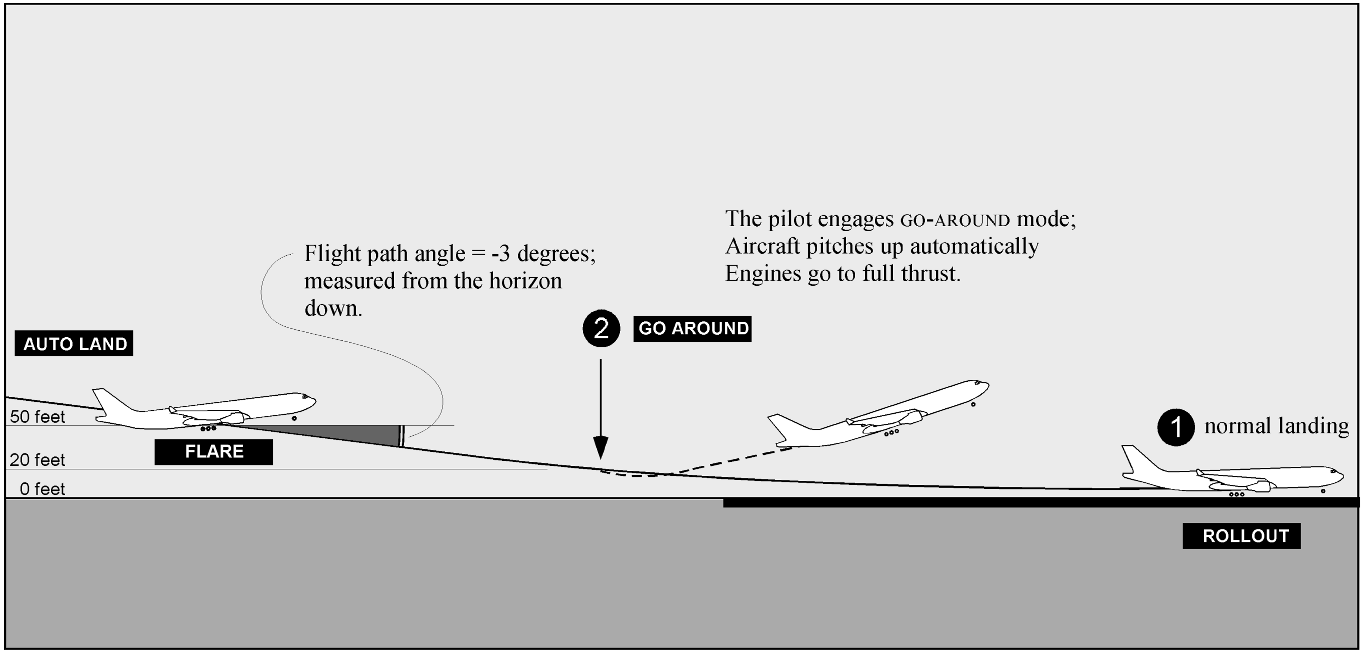
Flight path angle = -3 degrees;
measured from the horizon
down.
The pilot engages GO-AROUND mode;
Aircraft pitches up automatically
Engines go to full thrust.
AUTO LAND
2 GO AROUND
50 feet
FLARE
20 feet
0 feet
1 normal landing
ROLLOUT

Figure 17.3. Automatic landing and go-around (not to scale).

initiated any time during the approach to landing. Figure 17.3 shows a go-around that is initiated when the aircraft is at an altitude of 20 feet.

Verification of the Go-Around Maneuver

We want to verify the aircraft responses during a go-around maneuver. In particular, we wish to explore the consequence of engaging a go-around when the aircraft is close to the ground. Why? Because at that vulnerable point in time, the speed is at its lowest as the airplane is making the delicate transition from flying to landing. But how do we go about performing this verification? Very much as we did in the analysis of the intersection, we begin by considering the available options. There are two in this case: either the airplane lands or the airplane makes a go-around.

Unlike the intersection example, where car performance data, such as maximum braking and acceleration are readily available, information about AUTOLAND performance is proprietary information. This information is kept behind locked doors as a trade secret and is therefore unavailable publicly. For that reason, the information used in this analysis is based on aeronautical engineering textbooks and general knowledge of how modern autoland systems work. As usual, we are interested here in the approach and methodology for verification of an automatic system, and not so much in the details of a specific autoland system.

Safe Landing

The funnel-like shape in figure 17.4(a) shows us the region from which an autopilot can make a safe landing. There are three variables that combine to create this three-dimensional shape: the aircraft's altitude above the runway, the aircraft's speed, and the aircraft's flight-path angle (which is the angle at which the airplane descends toward the ground—see the inset at the top of figure 17.4[a]). In principle, the shape is computed in the same way the regions in the yellow light example were computed. We start from touchdown, where the flight path angle should be between 0 and -2 degrees and work our way back. (If the angle is greater than zero, the airplane will not be able to land; if the angle is less than -2 degrees, the aircraft's tail will hit the ground.) For each altitude, from 0 to 60 feet, we compute the speed and flight-path angle that the autopilot needs to maintain such that eventually the aircraft will make a safe landing.

This is exactly what we did in calculating the safe-braking region in the yellow interval dilemma. For example, at an altitude of 60 feet, if the flight-path angle is

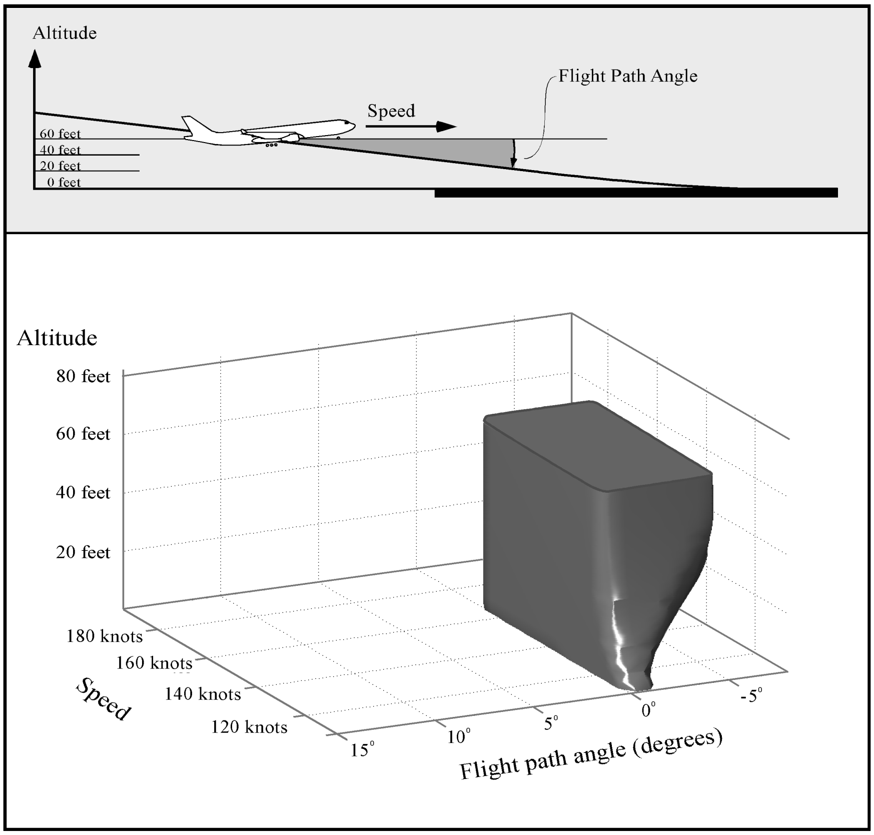

Figure 17.4 (a). Safe landing.

between 0 and -7 degrees and the aircraft's speed is between 120 and 170 knots, the autopilot will eventually make a safe landing. However, if the speed is below 100 knots the autopilot will not be able to make a safe landing.

Safe Go-Around

Figure 17.4(b) shows the safe region for executing a go-around. This region is rather large because unlike the safe landing region that funnels down to the runway with a tightly constrained angle and speed, the go-around can be executed safely at a variety of flight-path angles and speeds. But as you can see in figure 17.4(b), the shape has a wedge-like cutout at low speeds and negative flight-path angles. Why? Because it takes the go-around maneuver several seconds in order to pull the aircraft out of the descent and avoid hitting the ground. What figure 17.4(b) shows us is the combinations of altitudes, speeds, and flight-path angles from which a safe go-around is always guaranteed.

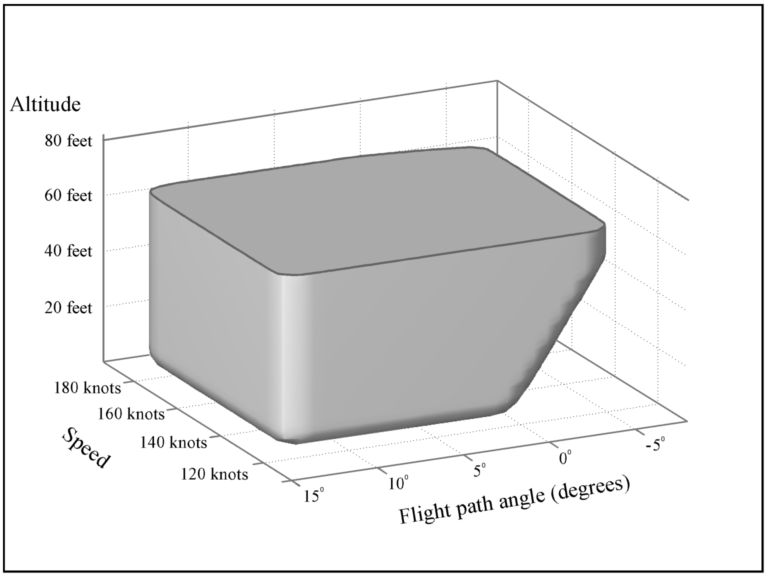

Figure 17.4 (b). Safe go-around.

We have identified both the safe region for landing and the safe region for go-around; now we want to verify that while the autopilot is controlling the airplane for a safe landing, the pilot can always break away from the landing and make a go-around to evade danger. To do this, we need to consider the safe "landing" region and the safe "go-around" region together. And for the sake of illustration, let us consider what will happen when the pilot engages the go-around mode and the aircraft is 20 feet above the ground. Figure 17.5(a) is a slice of the safe-landing region at 20 feet above the ground, figure 17.5(b) is the same for the safe go-around region, and figure 17.5(c) is the composite graph.

Composite Graph

The composite graph of figure 17.5(c) shows three emerging sub-regions: the light gray is the safe go-around region, the black is where safe go-around overlaps with safe landing, and the hatched is also part of the safe landing region. Because a go-around is usually triggered by a disturbance event that can occur at any time, the black region is where we always want to be: from here the autopilot can make a safe landing, and if a go-around is needed, the aircraft will be able to break away from the landing and escape safely.

So what about the hatched region? The hatched area is problematic, and here's why: under normal conditions the autopilot will try to make the landing when the flight-path angle is close to 0 degrees, but under less than normal

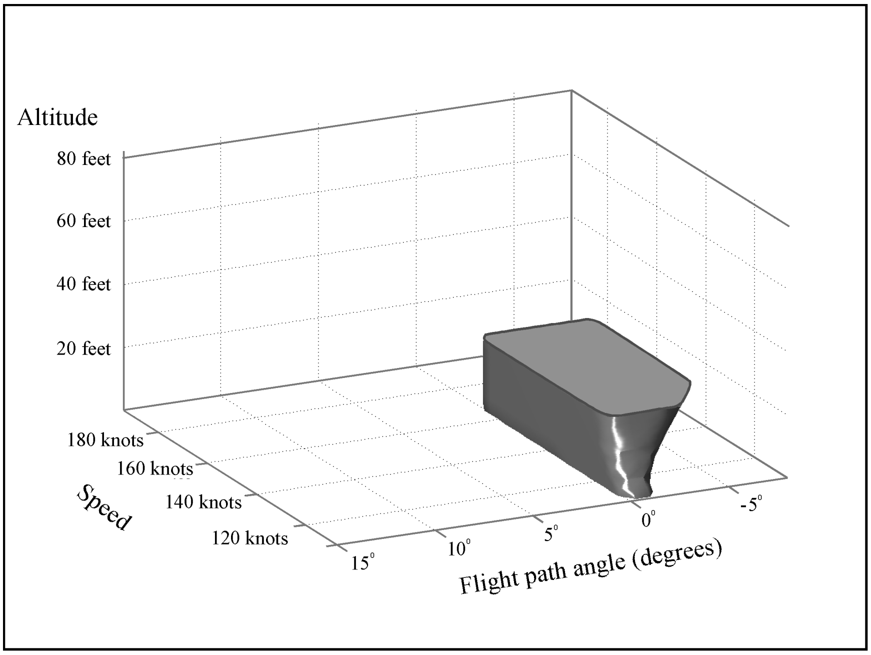

Figure 17.5 (a). Safe landing region at 20 feet.

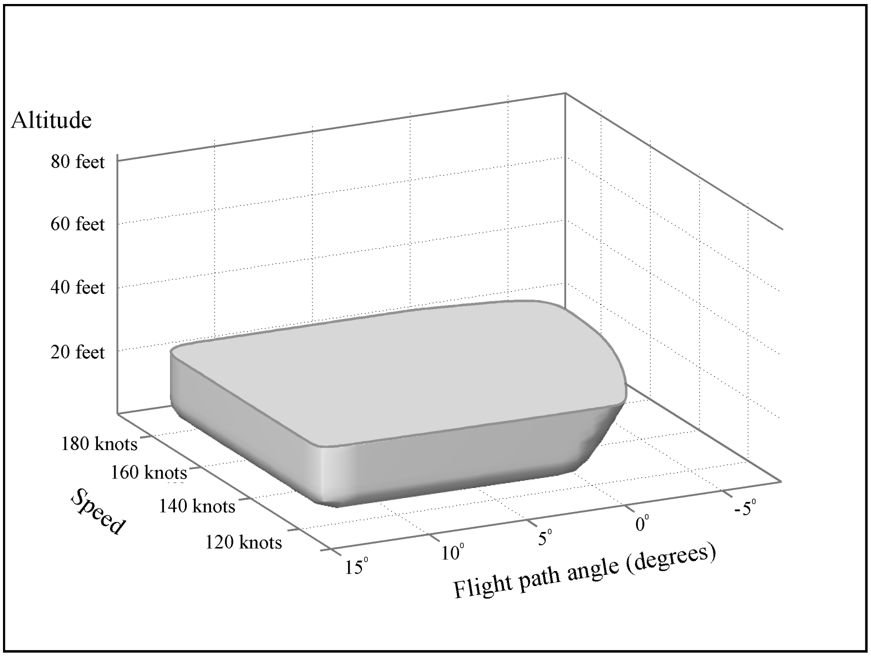

Figure 17.5 (b). Safe go-around at 20 feet.

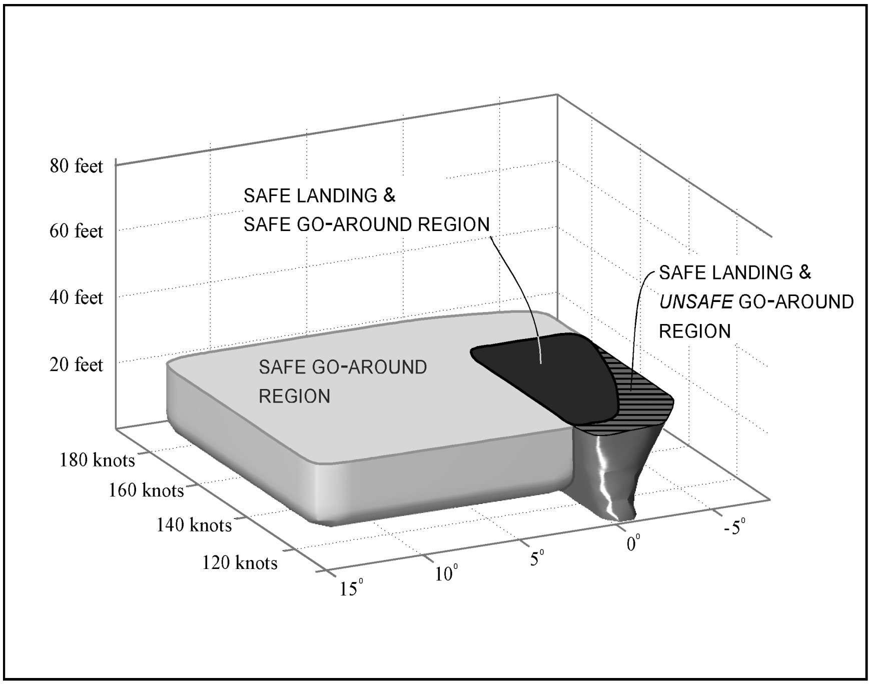

Figure 17.5 (c). Composite.

conditions, such as gusts or a strong tailwind, the autopilot may be operating in the hatched region. Nevertheless, the autopilot can handle the situation and still make a safe landing.

But we are not just concerned about safe landing; we also want to ensure a safe go-around. And here lies the problem: if the autopilot is operating in the hatched region, the aircraft will not be able to execute a safe go-around; neither from this altitude of 20 feet, nor from any lower altitude. In this example, if the autopilot is operating in the hatched region and a go-around is initiated, the aircraft may stall.

Automation Locks

There is a certain similarity between the yellow interval dilemma and the automatic landing example. In both cases, there are regions where the system is no longer safe; no matter what the driver or the autopilot does, they will not be able to escape the situation. In these regions the human-machine system finds itself in a lock.

But there also is an important difference between the two examples, which has to do with the locus of control: in the case of the yellow light, the driver is *manually* controlling the car. When we operate machines in everyday life, there are many situations in which we find ourselves in such a lock. Sometimes while driving on a highway the traffic is jammed and safe braking distances are hardly maintained. We all know that if someone ahead slams on the brakes there will be a chain accident, and because the lanes to the right and left are tightly packed, there is no escape. We are cognizant of this, yet we find ourselves again and again in these situations. There are many similar situations on the road, sometimes resulting from our actions or from inherent design deficiencies (e.g., poorly designed on-ramps). And we tend to accept the risks.

However, when it comes to an automated system, it is the machine itself that enters the potentially unsafe region from which recovery is difficult or impossible. Initially, the automation is in control, the user is monitoring the machine, and everything looks fine. And then, for example, air traffic control instructs the pilots to make a go-around. But when the user takes manual control or commands the automation to escape, the system is in a state from which recovery is difficult. This is an automation lock. Over the years, such "automation locks" have manifested themselves in several incidents and accidents.

China Airlines Boeing 747

On February 19, 1985, a China Airlines Boeing 747-SP, flying from Taipei, Taiwan, to Los Angeles, California, experienced an engine malfunction over the Pacific. (There are four engines on a Boeing 747 and the aircraft is designed to operate safely with three.) The aircraft was flying at 41,000 feet when the outmost right engine (number 4) got stuck in a low thrust setting. The aircraft began to slow down and the crew was trying to diagnose the problem and initiate remedial action.

Before and during the engine problem, the aircraft was on autopilot control. As soon as the outmost right engine malfunctioned, the aircraft started to roll to the right (due to asymmetric thrust—two good engines on the left side of the aircraft vs. only one and a half good engines on the right). But then the autopilot immediately countered by rolling the aircraft to the left. Meanwhile, the captain was focusing on trying to regain airspeed and was reluctant to disengage the autopilot and take manual control of the aircraft (as was recommended in his training and procedures for such a situation). A minute and a half later, the malfunctioning engine failed completely and was no longer producing any thrust. Now the autopilot had to work harder to roll the aircraft to the left in order to keep the wings level. The amount of left roll required to keep the aircraft flying level was coming close to the autopilot's control limits.

But inside the cockpit, except for the decreasing airspeed, the only feedback that would have depicted the worsening asymmetric situation and the autopilot's excessive attempts to counter the turn to the right, was the pilot's control wheel increasing left-wing-down deflection (about 6–10 degrees). However, this was an area that was not included in the captain's regular instrument scan, and since he was not "hands on" the wheel, he was not aware of this deflection.

The autopilot actions to counter the aircraft's tendency to roll to the right introduced a turning motion (side slip) and increased the aircraft's drag, which, in turn, further reduced the aircraft's speed. The captain reacted by switching the autopilot from the flight management computer control to a more simple pitch-hold mode and rotated a knob to begin a descent. Meanwhile, the autopilot was losing its struggle with the asymmetric thrust and the right wing was dropping. The aircraft began to gradually pitch down and descend while the autopilot was still fighting to bring up the right wing. Nevertheless, the autopilot did not provide any warning or indications to let the crew know of the alarming situation. Seconds later, the autopilot could no longer sustain the load of the wing and gave up—the right wing dropped rapidly, passed 45 degrees and the aircraft entered a downward spiral.

The large aircraft, with 251 passengers and 23 crewmembers onboard, plunged down out of control as the captain was trying to regain manual control of the aircraft and the flight engineer was attempting to restart the failed engine. The aircraft was in the clouds and this hindered the captain's ability to regain control of the aircraft. When the aircraft emerged from the clouds at 11,000 feet, the captain could see the horizon and was finally able to stop the dive and level the aircraft at 9,500 feet. The crew declared an emergency and made a safe landing at San Francisco Airport. One passenger and one crewmember suffered serious injuries during the 31,500-foot dive. The airplane suffered structural damage in the tail section.

It is important to note that at the time of the incident, the aircraft had already been 10 hours in the air crossing several time zones. The crew's ability to obtain, assimilate, and analyze the data from the flight displays and engine indicators as well as recall information from their training and procedures was probably impaired by the effect of monotony, boredom, and fatigue. In light of these factors, the captain's reluctance to disengage the autopilot and hand-fly the aircraft as soon as the engine malfunctioned was probably not because of over-trust in the automation per se, but perhaps because of his need to rely on the automation while being fatigued.

Simmons Airlines ATR-72

On October 31, 1994, a large commuter aircraft with 64 passengers, two flight attendants, and two pilots, was flying in bad weather to Chicago. The aircraft, flying at 16,000 feet, was in the clouds and had encountered icing conditions. As the aircraft approached Chicago's O'Hare airport, it was instructed to enter a holding pattern and maintain 10,000 feet. The crew knew they were flying in icing condition, but because the wings of the aircraft were far behind the cockpit, they could not visually see the accumulation of the ice on the wings. When ice builds up on an aircraft's wing, it changes the wing profile (or cross section) such that it is no longer smooth and efficient. The immediate implication is that the wing cannot produce as much lift as normal and the plane can therefore stall earlier.

The autopilot was engaged throughout the flight and when air traffic control instructed them to "descend and maintain 8,000 feet," the flight crew selected VERTICAL SPEED mode and began a descent to 8,000 feet. As the ice attached itself to the wings, it formed different shapes; specifically, the ice on the right wing of the aircraft was accumulating faster, causing the right wing to drop down, and rolling the aircraft into a right turn. Meanwhile, the autopilot was constantly countering (the right turn) by commanding a left turn. As this was going on, the crew had no idea of the deteriorating situation; they could not see the wing and the autopilot system did not indicate or alert them that it was working extremely hard to keep the aircraft's wings level. As the aircraft was descending through 9,500 feet, the autopilot could no longer compensate for the accumulation of ice on the right wing and counter the right roll. A second later the autopilot disengaged. This was by design, because there are conditions in the autopilot logic that trigger an automatic disengagement when the autopilot is working beyond a pre-determined operating range.

When the autopilot disengaged and quit, the right wing, no longer held up by the autopilot, immediately dipped down, corkscrewing the aircraft into a brutal right turn. The crew, who just seconds before had been sitting passively and monitoring an aircraft flying on autopilot, all of a sudden had a very vicious and strangely behaving aircraft on their hands; the accumulation of ice on the wings dramatically changed the way the aircraft behaved (and altered its handling characteristics). The nose dropped down to 15 degrees below the horizon; and the aircraft rolled rapidly to the right—the right wing reaching 77 degrees down. From there on the crew constantly tried to gain control over the aircraft. But the aircraft was rolling to the right and pitching the nose down to 60 degrees below the horizon. The aircraft was losing altitude fast. Twenty-seven seconds after the autopilot disengaged, the aircraft crashed in a soybean field.

The National Transportation Safety Board and most aviation experts agree that the flight crew's actions, before and after the autopilot quit, were consistent with their training and knowledge and in line with what is expected from a competent crew. It was extremely difficult—given the abruptness and severity of the unexpected roll and the changing characteristics of the aircraft (due to ice accumulation)—for the pilots to recover from this icing encounter and subsequent loss of control.

In Conclusion

In both of the above cases, the autopilot did not clearly indicate to the pilots the increasingly troublesome situation, nor did it give a warning that it was about to give up. It just did. As a consequence, the flight crews found themselves in situations from which recovery was difficult and at times impossible. Moreover, the immediate and unexpected transition from being a passive observer (monitoring the autopilot, sometimes for hours) to actively controlling a disabled aircraft that behaves unpredictably, was extremely difficult for the pilots. Dealing with these adverse situations and regaining control required split-second responses—a tall order, in particular when there was no time to ascertain the current status of the aircraft and assess the situation.

With increased automation and the shift of authority toward the machine, such "automation locks" pose serious threats to safe and reliable operation. They also highlight an important aspect of human-automation interaction, which has to do with the way we perceive automated machines. We, as humans, have a hard time accepting the possibility that an automated machine that is given full control will blindly enter into an unsafe and potentially disastrous situation. These automation locks, which exist in modern-day systems, bring us again face-to-face with HAL. Just as HAL, the ever-reliable machine, locked the astronauts out of control in *2001: A Space Odyssey*—in these two examples the autopilot, usually very reliable, locked the pilots into a situation from which recovery was extremely difficult.

While I do not wish to imply that it is necessarily the case that human operators can always recover from failures that automation can not, sometimes this is indeed the case. Had the pilots in the above mishaps been informed in time of the impending problems, they could have manually stabilized the aircraft and most likely recovered. Furthermore, the automation is usually designed to control the system under pre-specified operating conditions that do not include all irregular circumstances that might occur as a result of environmental impediments or internal failures. These impediments and failures are frequently too many and too diverse to account for and to guard against. The human operator, if suitably informed, could decide whether to continue and rely on the automation, or assume control manually. Thus, suitable indications and annunciations regarding departures from normal operating conditions are extremely important when designing automatic controls of safety-critical systems.

Before closing, it is important to mention that automation locks can be addressed in several ways: one, by identifying those critical situations in which the user will not be able to bring the automated system back to safety; two, by improving the design and thereby reducing the size of unsafe regions; three, by developing more sophisticated control systems (for example, the autopilot) that will resist entry into unsafe regions; and most importantly, by providing the user with timely and appropriate feedback about the dynamic behavior of the control system and departures from normal operating conditions. We need interfaces that know when to abstract-away information and when to display additional information—and how to do this in a way that adapts to the user's needs, interaction style, and the context of the situation. Developing such "intelligent" interfaces requires a level of sophistication that does not exist today. In many ways, interface technology lags behind advances in information technology and the ever-powerful machine. Needless to say, there are many challenges ahead when it comes to interface design and human-automation interaction.

then decide to increase speed or shift down (either manually, or by letting the transmission do it automatically).

The autopilot example presented in this chapter is based on an article by Asaf Degani and Michael Heymann entitled "Formal Verification of Human-Automation Interaction," which appeared in the *Human Factors Journal* (Volume 44, issue 1, pages 28-43). The *Wall Street Journal*, in an article titled "Cold, Hot, Cold, Hot: Employees Only Think they Have Control" (January, 15, 2003, page B-1), reports that Heating Ventilation and Air Conditioning (HVAC) technicians admit to what office workers suspected for years: that many office thermostats are fake. The estimates for the percentage of "fixed" thermostats ranges from 2 percent according to one technician to 90 percent according to another. Such thermo-fraud has been going for some 40 years—only increasing when utility prices go up.

Beyond just verification of a given (e.g., prototype), it is possible to extend the concepts of error, augmenting, and restricting states to a systematic methodology for generating correct interfaces. That is, a process that takes into account the machine's behavior and the user's task requirements and generates an interface solution. Further, it is possible to use this process not only to generate a correct interface, but also one that is the simplest possible. Such a methodology exists, and we have already used it in this book. In particular, the methodology and algorithm for generating correct and succinct interfaces follows the same steps that we took in redesigning the cruise-control system in chapter 11. (For a detailed discussion on this topic see Michael Heymann and Asaf Degani, *On Abstractions and Simplifications in the Design of Human-Automation Interfaces*, NASA Technical Memorandum number 211397, which was published in 2002). For a shorter and lighter treatment of this topic, see Asaf Degani and Michael Heymann, "Analysis and Verification of Human-Automation Interfaces," which appeared in the Proceedings of the 10th International Conference on Human-Computer Interaction, June 22-27, 2003.

Chapter 17

The examples, analysis, and methods presented in this chapter are based on a recent report on analysis and verification of automation systems and interfaces (see Meeko Oishi, Claire Tomlin, and Asaf Degani, *Safety Verification of Hybrid Systems: Applications for User Interface Design*; NASA Technical Memorandum, 2003). As for the yellow light interval, see an article titled "A Review of the Yellow Interval dilemma," by C. Liu, R. Herman, and D. Gazis, which appeared in a journal called *Transportation Research A*, volume 30, issue number 5, 1996. With respect to entering an intersection, the California Vehicle Code (of 2002) states the following rules (which I edited here for comprehension):

- A driver facing a yellow light is, by that signal, warned that the green light period is ending or that a red light will be shown immediately thereafter (regulation 21452(a)).
- A driver facing a red light alone shall stop at a marked limit line (regulation 21453(a))

Since the regulation says nothing about entering the intersection on yellow and encountering a red while the car is inside, the situation is open for interpretation. The common interpretation used by police officers and judges is that you can enter the intersection when the light is yellow (regulation 21452(a) does not prohibit entering—it only says that you are warned of an impending red) and therefore it is okay if the light changes to red while you are inside the intersection.

Many of the examples discussed in this book have distinct, or discrete, modes as well as continuous parameters (reference values). Such systems, which have both discrete and continuous dynamics, are called *hybrid systems*. In the last decade or so, a theory and methodologies for analysis and verification of such systems have been developed. The theory extends the classical notion of a finite-state machine to add continuous dynamics such as time, speed, pressure, and more. These methods are used in the analysis of systems that have complicated dynamics, such as collision avoidance systems for aircrafts and also for cars. As shown in this chapter, these methods can also be extended to the analysis and verification of human-automation interaction and interface methods.

There is an implicit assumption, especially when it comes to designing interfaces, that the underlying system works per specifications. Assuming that the underlying system behaves as specified is imperative for the analysis and verification of interfaces, because otherwise it is impossible to analyze an interface and come to any meaningful conclusions. Therefore, one of the objectives of this chapter was to highlight the fact that interface analysis and verification can *only* take place after there is an assurance that the underlying system behaves as specified and all the unsafe regions are understood and accounted for. In other words, the model of the system must be sound.

To further illustrate this point, say that you are asked to develop a new system for aiding drivers. Specifically, you want to provide guidance to the driver about whether he or she should stop at a yellow light or proceed through an intersection. You can think about a display on the dashboard that will indicate GO if the car is in the safe "proceed through" region and STOP if in the "safe braking" region. But if at certain speeds and distances from the intersection, the driver will commit a violation regardless of whether he or she stops or goes, then, of course, the interface is useless.

But even if you perform the above analysis and identify the unsafe region, just indicating it to the driver is not enough. Why? Because it's too late! In many respects it is like telling a diver who has already jumped off the platform that there is no water in the pool. I hope you'll recognize that in these situations the problem is beyond the interface. The only way to deal with the problem is to provide an advisory system that monitors the car's speed and distance with respect to the unsafe region. When the driver is *about* to enter the unsafe region, the system can warn the driver to change speed and avoid entry.

For a more detailed description of the Boeing 747 incident, see *China Airlines Boeing 747-SP, 300 nautical miles northwest of San Francisco, California; February 19, 1985*. This accident report was published in 1986 by the National Transportation Safety Board (NTSB report number AAR-86/03). I obtained the information regarding the commuter aircraft accident from the NTSB accident report (*In-flight icing encounter and loss of control, Simmons Airlines, doing business as American Eagle Flight 4184. Roselawn Indiana, October 31, 1994*. NTSB Report number AAR-96/01) as well as several trade publications on this accident.

Chapter 18

The Luddites claimed to be led by Ned Ludd, also known as "King Ludd" and "General Ned Ludd." It is believed that he was responsible for the destruction of two large knitting machines that produced inexpensive stockings, and his signature does appear on a "workers manifesto" of the time. Whether or not Ludd actually existed is unclear; it is also well documented that acts of machine wrecking took place prior to Ludd's appearance on the scene.